

On the Space Complexity of Set Agreement^{*}

Carole Delporte-Gallet[†] Hugues Fauconnier[†]

Petr Kuznetsov[‡] Eric Ruppert[§]

Abstract

The k -set agreement problem is a generalization of the classical consensus problem in which processes are permitted to output up to k different input values. In a system of n processes, an m -obstruction-free solution to the problem requires termination only in executions where the number of processes taking steps is eventually bounded by m . This family of progress conditions generalizes wait-freedom ($m = n$) and obstruction-freedom ($m = 1$). In this paper, we prove upper and lower bounds on the number of registers required to solve m -obstruction-free k -set agreement, considering both one-shot and repeated formulations. In particular, we show that repeated k set agreement can be solved using $n + 2m - k$ registers and establish a nearly matching lower bound of $n + m - k$.

1 Introduction

Algorithms that allow processes to reach agreement are one of the central concerns of the theory of distributed computing, since some kind of agreement underlies many tasks that require processes to coordinate with one another. In the classical consensus problem, each process begins with an input value, and all processes must agree to output one of those input values. Chaudhuri [3] introduced the k -set agreement problem, which generalizes the consensus problem by allowing processes to output up to k different input values in any execution. Consensus is the special case where $k = 1$. Set agreement is trivial for n processes if $k \geq n$: each process can simply output its own input value.

We consider the k -set agreement problem for $k < n$ in an asynchronous system equipped with shared read/write registers. To satisfy *wait-free* termination, non-faulty processes must terminate even if an arbitrary number of processes fail. The impossibility of solving wait-free k -set agreement using registers was a landmark result proved by three groups of researchers [2, 10, 11]. However, Herlihy, Luchangco and Moir [9] observed that k -set agreement *is* solvable (even for $k = 1$) under a weaker termination property, known as *obstruction-freedom* or *solo-termination*, which requires that a process must eventually terminate if it takes enough steps without interruption from other

^{*}The first two authors were supported by the Agence Nationale de la Recherche, project DISPLEXITY, the third author was supported by the Agence Nationale de la Recherche, under grant agreement ANR-14-CE35-0010-01, project DISCMAT, and the fourth author by the Fondation Sciences Mathématiques de Paris and the Natural Sciences and Engineering Research Council of Canada.

[†]LIAFA, Université Paris-Diderot

[‡]Télécom ParisTech

[§]York University

processes. Obstruction-freedom was introduced as a way of separating concerns: obstruction-free algorithms maintain safety properties in all possible executions, but make progress only when one process can run for long enough without encountering contention. Various scheduling mechanisms designed to reduce contention (such as backing off) can then be used to satisfy this condition.

Taubenfeld [12] introduced the m -obstruction-freedom progress property, which requires that, in any execution where at most m processes take infinitely many steps, each process that continues to take steps will eventually terminate successfully. Wait-freedom and obstruction-freedom are special cases, with the extreme values $m = n$ and $m = 1$, respectively. Like ordinary obstruction-freedom, m -obstruction-free algorithms guarantee safety in all runs. However, m -obstruction-freedom provides a stronger progress property: larger values of m require less rigid constraints on the scheduler in order to ensure progress. Since k -set agreement has no wait-free solution among $k + 1$ processes, it follows that there is no m -obstruction free solution when $m > k$. The converse follows from the work of Yang, Neiger and Gafni [14]: m -obstruction-free k -set agreement *can* be solved if $m \leq k$. In this paper, we study how the number of registers required to solve m -obstruction-free k -set agreement among n processes depends on the parameters m, k and n .

Previously, the only non-trivial space lower bound was for the very special case where $m = k = 1$. In this case, Fich, Herlihy and Shavit [6] showed $\Omega(\sqrt{n})$ registers are needed. The best upper bound for this case is the trivial one of n registers, which comes from the fact that n (large) single-writer registers can implement any number of multi-writer registers [13]. Closing the gap between the linear upper bound and the $\Omega(\sqrt{n})$ lower bound is a major open problem. Unfortunately, there has been no progress on this gap in the past two decades.

We first prove nearly tight linear upper and lower bounds on the number of registers required for *repeated* set agreement. In many applications, such as Herlihy’s universal construction [8], there is a sequence of (independent) agreement tasks that must be solved, rather than just one. We define the repeated k -set agreement problem to model this situation. Processes access an infinite sequence of instances k -set agreement in order. For all executions and for all i , processes accessing the i th instance of k -set agreement may output at most k of the values that are used as inputs to that instance.

We prove that any m -obstruction-free solution to repeated k -set agreement among n processes requires at least $n + m - k$ registers. We also give a novel algorithm for this task using $\min(n + 2m - k, n)$ registers. Previously, the only known set agreement algorithm that uses fewer than n registers was a 1-obstruction-free k -set agreement algorithm that uses $2n - 2k$ registers [4]. Our algorithm generalizes that algorithm (to handle any value of m) and improves the number of registers used in the case where $m = 1$ from $2(n - k)$ to $n - k + 2$. For the case where $m = k = 1$, our results establish that obstruction-free repeated consensus requires exactly n registers. Thus, the gap between the $\Omega(\sqrt{n})$ lower bound and the $O(n)$ upper bound is closed when we consider the *repeated* version of the problem.

For the one-shot version of k -set agreement, we focus on the restricted case of anonymous systems, where processes do not have unique identifiers and are all programmed identically. We prove that any anonymous algorithm must use more than $\sqrt{m(\frac{n}{k} - 2)}$ registers. The $\Omega(\sqrt{n})$ lower bound of Fich, Herlihy and Shavit [6] (for the anonymous case) is a special case of our result with $m = k = 1$, but the new result gives additional insight into the problem by showing a dependence on m and k . Moreover, the technique used in our proof is somewhat different, since it requires building an execution involving many different input values where each process is prevented from learning about any input value different from its own. We also prove that it is possible

| | Repeated | One-shot |
|---------------|--|---|
| Non-Anonymous | lower: $n + m - k$ (Section 3) upper: $\min(n + 2m - k, n)$ (Section 4) | lower: 2 [4] upper: $\min(n + 2m - k, n)$ (Section 4) |
| Anonymous | lower: $n + m - k$ (Section 3) upper: $(m + 1)(n - k) + m^2 + 1$ (Section 6) | lower: $\sqrt{m(\frac{n}{k} - 2)}$ for $D = \mathbb{N}$ (Section 5) upper: $(m + 1)(n - k) + m^2$ (Section 6) |

Figure 1: Lower and upper bounds on the number of registers to solve m -obstruction-free k -set agreement among n processes, where $1 \leq m \leq k < n$ and input values are from domain D (with $|D| > k$). Our main results appear in boldface; the others are corollaries.

to solve the problem anonymously. Our algorithm for the repeated version of the problem uses $(m + 1)(n - k) + m^2 + 1$ registers. (The usual construction using n single-writer registers is not applicable, since it presupposes unique identifiers.)

Figure 1 summarizes our results. Our four main results are in boldface; the others are corollaries.

2 Preliminaries

We consider the standard asynchronous shared-memory model, in which $n > 1$ processes p_1, \dots, p_n communicate by applying read and write operations to shared *registers*. The registers are *multi-writer* and *multi-reader*, i.e., there are no restrictions on which processes may access which registers.

Each process has a local state that consists of the values stored in its local variables and a programme counter. A computation of the system proceeds in *steps* performed by the processes. Each step is one of the following: (1) an invocation of an operation, (2) a read or write operation on a shared register, (3) local computation that results in a change of a process’s state, or (4) a response of an operation. Writes update the state of a shared register. All steps may update the local state of the process that performs it. A *configuration* specifies the state of each register and the local state of each process at one moment. In an *initial configuration*, all registers have the initial values specified by the algorithm and all processes are in their initial states.

A process is *active* if an operation has been invoked on the process but the operation has not yet produced a matching response; otherwise the process is called *idle*. We assume that an operation can only be invoked on an idle process and only active processes take steps. We focus on deterministic algorithms. Thus, given the current local state of an active process, the algorithm for this process stipulates the unique next *step* the process can perform. An *execution fragment* of an algorithm is a (possibly infinite) sequence of steps starting from some configuration that “respects” the algorithm for each process. An *execution* is an execution fragment that starts from the initial configuration. An operation is completed if its invocation is followed by a matching response. In an infinite execution, a process is *correct* if it takes an infinite number of steps or is idle from some point on.

Our algorithms make use of multi-writer *snapshot objects* [1], which can be implemented from registers. A snapshot object stores a vector of r values and provide two atomic operations:

$update(i, v)$ ($i \in \{1, \dots, r\}$), which writes value v to component i , and $scan()$, which returns the vector of the most recently written values to components $1, \dots, r$.

2.1 Set agreement

We begin with a formal definition of the *repeated k -set agreement* problem. Processes may perform $PROPOSE(v)$ operations, where v is drawn from an input domain D . Each $PROPOSE$ operation outputs a response from D when it terminates. For an execution α , let $In_i(\alpha)$ be the set of values that are used as the argument to some process's i th invocation of $PROPOSE$ and let $Out_i(\alpha)$ be the set of values that are the response of some process's i th $PROPOSE$ operation. Then, in every execution α of an algorithm that solves repeated k -set agreement the following properties must hold.

- *Validity*: $\forall i, Out_i(\alpha) \subseteq In_i(\alpha)$.
- *k -Agreement*: $\forall i, |Out_i(\alpha)| \leq k$.

An m -obstruction-free algorithm must additionally satisfy the following termination condition.

- *m -Obstruction-Freedom*: in every execution in which at most m processes take infinitely many steps, every correct process completes each of its operations.

The special case when $k = 1$ is called *consensus*. In the *(one-shot) k -set agreement problem*, every process invokes $PROPOSE$ at most once.

It is known that wait-free $(k + 1)$ -process k -set agreement cannot be solved using registers [2, 10, 11]. This implies the following lemma, which we shall use to prove our space lower bounds.

Lemma 1. *Let A be any algorithm that solves m -obstruction-free k -set agreement using registers. For any set V of m input values and any set Q of m processes, there is an execution of A in which only processes in Q take steps and all values in V are output.*

Proof. Suppose the opposite for some sets V and Q and consider all executions of A in which only processes in Q with inputs in V take steps. By the assumption, at most $m - 1$ distinct values are decided in each of these executions, which implies a wait-free m -process $(m - 1)$ -set agreement algorithm, violating [2, 10, 11]. \square

Lemma 1 implies that no algorithm can solve m -obstruction-free k -set agreement using registers if $k < m$. In the rest of the paper, we derive lower and upper bounds on the space complexity of m -obstruction-free k -set agreement for n processes, where $m \leq k < n$. (If $k \geq n$, the problem is trivial and no registers are required: each process can simply output its own input value.)

3 Lower Bound for Repeated Set Agreement

In this section, we prove that solving m -obstruction-free repeated k -set agreement among n processes requires at least $n + m - k$ registers. Since the proof is technical, we first provide a brief overview. For simplicity, assume for now that $k + 1$ is a multiple of m . We assume that there is an algorithm that uses fewer than $n + m - k$ registers, and construct an execution in which processes return $k + 1$ different values in some instance of set agreement, contradicting the k -agreement

property. The proof first constructs $c = \frac{k+1}{m}$ disjoint sets Q_1, Q_2, \dots, Q_c of m processes each, and an execution α that passes through a sequence of configurations D_1, D_2, \dots, D_c with the following property. For $1 \leq i < c$, *every possible* execution fragment by the processes in Q_i starting from D_i writes only to registers that are overwritten immediately after D_i in α . Moreover, processes in Q_i take no more steps after D_i in α . We can then splice into α any execution fragment by processes in Q_i at D_i , knowing that the rest of α will not be affected, since all evidence of the inserted steps will be overwritten. For each group Q_i , the fragment we splice into α accesses a “fresh” instance of set agreement that was never accessed during α . (In each fragment that is spliced in, only the m processes in Q_i take steps, so all PROPOSE operations terminate and the processes will eventually reach and complete the fresh instance of set agreement.) We ensure that these groups of m processes output disjoint sets of m different values each for this one instance of set agreement, for a total of $c \cdot m = k + 1$ different outputs, a contradiction.

Theorem 2. *Any algorithm for m -obstruction-free repeated k -set agreement among n processes requires at least $n + m - k$ registers.*

Proof. To derive a contradiction, assume there exists an algorithm for m -obstruction-free repeated k -set agreement using $r = n + m - k - 1$ registers. Let $c = \lceil \frac{k+1}{m} \rceil$. Since $k \geq m$, we have $c \geq 2$. We define a *block write* to a set A of registers by a set P of processes to be an execution fragment in which each process of P takes a single step, such that the set of registers written during the fragment is A .

We first construct an execution

$$C_0 \xrightarrow{\alpha_1} D_1 \xrightarrow{\beta_1} C_1 \xrightarrow{\alpha_2} D_2 \xrightarrow{\beta_2} C_2 \xrightarrow{\alpha_3} \dots \xrightarrow{\beta_{c-1}} C_{c-1} \quad (1)$$

and sets A_1, \dots, A_{c-1} of registers such that C_0 is the initial configuration and for all j ,

1. α_j is an execution fragment containing only steps by two disjoint sets P_j and Q_j of processes that goes from configuration C_{j-1} to configuration D_j ,
2. β_j is a block write to A_j by P_j that goes from configuration D_j to configuration C_j ,
3. $|Q_1| = k + 1 - (c - 1)m$,
4. if $j > 1$, $|Q_j| = m$,
5. $|P_j| = |A_j|$,
6. $Q_j \cap Q_{j'} = \emptyset$ for $j' \neq j$,
7. $Q_j \cap P_{j'} = \emptyset$ for $j' > j$, and
8. there is no execution fragment starting from D_j in which only processes in Q_j take steps and some process writes outside A_j .

BASE CASE ($j = 0$): Let C_0 be the initial configuration.

INDUCTIVE STEP: Let $1 \leq j \leq c - 1$. Assume we have constructed the execution from C_0 to C_{j-1} satisfying all the properties. The algorithm in Figure 2 constructs the execution fragment α_j and the sets P_j , Q_j and A_j . Then, let β_j be the execution fragment starting from D_j where each process in P_j takes a single step and let C_j be the resulting configuration.

```

1  let  $\alpha_j$  be the empty execution fragment
2   $D_j \leftarrow C_{j-1}$ 
3   $P_j \leftarrow \emptyset$ 
4   $A_j \leftarrow \emptyset$ 
5  if  $j > 1$  then  $size \leftarrow m$  else  $size \leftarrow k + 1 - (c - 1)m$ 
6  let  $Q_j$  be a set of  $size$  processes disjoint from  $Q_1 \cup Q_2 \cup \dots \cup Q_{j-1}$ 
7  loop until no execution fragment starting from  $D_j$  by  $Q_j$  writes outside  $A_j$ 
8      let  $\delta$  be an execution fragment starting from  $D_j$  by  $Q_j$  until some process  $q \in Q_j$  is poised for
          the first time to write to a register that is not in  $A_j$  and let  $R$  be that register
9       $\alpha_j \leftarrow \alpha_j \cdot \delta$ 
10     let  $D_j$  be the configuration reached from  $C_{j-1}$  by performing  $\alpha_j$ 
11     let  $q'$  be some process outside  $Q_1 \cup Q_2 \cup \dots \cup Q_j \cup P_j$ 
12      $A_j \leftarrow A_j \cup \{R\}$ 
13      $P_j \leftarrow P_j \cup \{q\}$ 
14      $Q_j \leftarrow (Q_j - \{q\}) \cup \{q'\}$ 
15 end loop
16 output  $\alpha_j, D_j, P_j, Q_j, A_j$ 

```

Figure 2: Algorithm used in the proof of Theorem 2 to construct α_j, D_j, P_j, Q_j and A_j .

Observe that the construction algorithm terminates: each loop iteration adds a new register to A_j , so it terminates after at most r iterations. We next check that the required processes on line 0 and 0 exist. When $j = 1$, we have $size = k + 1 - (c - 1)m = k + 1 - \lceil \frac{k+1}{m} \rceil \cdot m + m \leq m < n$, so one can choose the required processes on line 0. For $j > 1$, one can choose the process on line 0 because

$$\begin{aligned}
|Q_1 \cup \dots \cup Q_{j-1}| &= k + 1 - (c - 1)m + (j - 2)m \\
&\quad \text{(by induction hypothesis 3, 4 and 6)} \\
&\leq k + 1 - (c - 1)m + (c - 3)m \\
&\quad \text{(since } j \leq c - 1\text{)} \\
&= k + 1 - 2m \leq n - 2m \\
&\quad \text{(since } k < n\text{).}
\end{aligned}$$

Similarly, one can choose the required process q' at line 0 because

$$\begin{aligned}
&|Q_1 \cup \dots \cup Q_j \cup P_j| \\
&\leq k + 1 - 2m + |Q_j| + |P_j| \\
&\quad \text{(since } |Q_1 \cup \dots \cup Q_{j-1}| \leq k + 1 - 2m\text{)} \\
&\leq k + 1 - m + r - 1 \\
&\quad \text{(since } |Q_j| = m \text{ and } |P_j| = |A_j| \leq r - 1\text{)} \\
&= n - 1 \\
&\quad \text{(since } r = n + m - k - 1\text{).}
\end{aligned}$$

We verify the construction satisfies all of the properties. Line 0 of the algorithm updates D_j each time α_j is updated, to ensure property 1. Property 2 is true by definition of β_j and C_j . Q_j is

initialized to a set whose size satisfies property 3 or 4 on line 0 and the size of this set is preserved whenever Q_j is altered on line 0. P_j and A_j are initialized to be empty, and both are updated by adding one element to each on line 0 and 0, so they remain the same size after every iteration of the loop. (Note that P_j and Q_j are disjoint at the beginning of each iteration of the loop, so line 0 does add a new process to P_j .) Every process placed in Q_j at line 0 or 0 was chosen to be outside $Q_1 \cup \dots \cup Q_{j-1}$, guaranteeing property 6. Similarly, processes added to P_j are always outside $Q_1 \cup \dots \cup Q_{j-1}$, and whenever a process is added to P_j , it is removed from Q_j , so property 7 is satisfied. Finally, property 8 is guaranteed by the exit condition of the loop. This completes the inductive construction.

Now, let s be the maximum number of invocations of PROPOSE by any process in the execution that takes the system to configuration C_{c-1} . Let Q_c be a set of m processes disjoint from $Q_1 \cup \dots \cup Q_{c-1}$. (These m processes exist since $|Q_1 \cup \dots \cup Q_{c-1}| = k + 1 - m \leq n - m$.) Let $D_c = C_{c-1}$.

For each $j \in \{1, \dots, c\}$, we now construct an execution fragment γ_j by the processes in Q_j starting from D_j . Since $|Q_j| \leq m$, each PROPOSE in γ_j must terminate. First, the processes in Q_j run one by one until each completes its first s invocations of PROPOSE. Then, the processes of Q_j run their $(s + 1)$ th invocation of PROPOSE, each using its own id as its input value so that they decide $|Q_j|$ different output values. By Lemma 1, such an execution fragment exists. Note that for $j < c$, γ_j cannot write outside of A_j , by property 8. So, all traces of γ_j 's activity are obliterated by the block write β_j . Thus, we can insert $\gamma_1, \dots, \gamma_c$ into execution (1) at D_1, \dots, D_c , respectively, and the resulting execution is still legal. In the resulting execution, the number of distinct outputs for the $(s + 1)$ th instance of set agreement is $\sum_{j=1}^c |Q_j| = k + 1$, violating k -agreement. This completes the proof. \square

4 Algorithm for Repeated Set Agreement

4.1 One-shot k -set agreement

We first give an algorithm that uses a snapshot object of $r = n + 2m - k$ components to solve (one-shot) m -obstruction-free k -set agreement, and then describe how to extend it to solve repeated set agreement. The one-shot algorithm is shown in Figure 3. Roughly speaking, the first $k - m$ processes to decide can output arbitrary values, but we ensure that the last $\ell = n - k + m$ processes all agree on at most m different values (for a total of at most k different values).

Each process stores its preferred value in its local variable *pref*. Initially, it prefers its own input value. Each process executes a loop in which it stores its *pref* and identifier into a component of the snapshot object, takes a scan of the snapshot object and updates its *pref* variable based on the scan. The location i that the process updates advances in each iteration of the loop, as long as the process's *pref* value remains the same. When the process updates its *pref*, it does not advance to the next location: instead it updates the same location during the next iteration of the loop.

The process repeats this loop until a scan returns a vector containing at most m different value-id pairs, at which point it returns one of those values. In each iteration, a process updates its *pref* value when it does not see any copies of its current value-id pair anywhere in the vector returned by the scan, except for the component it just updated, *and* it does see two copies of some other pair. In this case, it adopts the value of the pair that appears twice as its *pref*.

```

1  Shared variable:
2    A: snapshot object with  $r = n + 2m - k$  components, each initially  $\perp$ 
3  PROPOSE( $v$ )
4     $pref \leftarrow v$ 
5     $i \leftarrow 0$ 
6    loop
7      update  $i$ th component of  $A$  with  $(pref, id)$ 
8       $s \leftarrow \text{scan of } A$ 
9      if  $|\{s[j] : 0 \leq j < r\}| \leq m$  and  $\forall j, s[j] \neq \perp$  then
10         let  $j_1 \leftarrow \min\{j_1 : \exists j_2 > j_1 \text{ such that } s[j_1] = s[j_2]\}$ , output value in  $s[j_1]$  and halt
11         if  $\forall j \neq i, s[j] \notin \{\perp, (pref, id)\}$  and  $\exists j_1 \neq j_2$  such that  $s[j_1] = s[j_2]$  then
12              $j_1 \leftarrow \min\{j_1 : \exists j_2 > j_1 \text{ such that } s[j_1] = s[j_2]\}$ 
13              $pref \leftarrow \text{value in } s[j_1]$ 
14         else  $i \leftarrow (i + 1) \bmod r$ 
15     end loop
16 end PROPOSE

```

Figure 3: Algorithm for m -obstruction-free k -set agreement. Code for a process with identifier id .

The algorithm in Figure 3 is an improvement on the algorithm of [4], which was designed for the special case where $m = 1$ and uses $2(n - k)$ registers, compared to the $n - k + 2$ registers used by ours.

We now prove that the algorithm in Figure 3 indeed solves m -obstruction-free k -set agreement. It is easy to see that **validity** holds: the only values that can appear in the snapshot object or in a process's local $pref$ variable are input values. Thus, only input values can be produced as outputs.

Before proving k -agreement and termination, we first establish the following invariant.

Lemma 3. *For each process identifier id , all the pairs in A with identifier id have the same value.*

Proof. To derive a contradiction, assume there is an execution that reaches a configuration C in which $A[i_1] = (v_1, id)$ and $A[i_2] = (v_2, id)$ where $v_1 \neq v_2$. Let p_{id} be the process with identifier id . Let u_1 and u_2 be the last steps before C in which p_{id} updates $A[i_1]$ and $A[i_2]$, respectively. Without loss of generality, assume u_1 is before u_2 . Then, between u_1 and u_2 , p_{id} changes its $pref$ variable at line 0. Consider the first time after u_1 that p_{id} performs such a change, and let i^* and s^* be the values of p_{id} 's local variables i and s at that time. Since s^* was obtained from a scan between u_1 and C and $A[i_1] = (v_1, id)$ throughout that interval, $s^*[i_1]$ is (v_1, id) . Thus, $i^* = i_1$; otherwise the test on line 0 would not be satisfied, and p_{id} would not change $pref$ at line 0. Therefore, in the next iteration of the loop, p_{id} will update location $A[i_1]$. This update is after u_1 and no later than u_2 (and hence before C), which contradicts the definition of u_1 as the last update performed by p_{id} on $A[i_1]$ before C . \square

To prove k -agreement, let $\ell = n - k + m$. If at most $n - \ell$ processes decide, then k -agreement is trivial since $n - \ell = k - m < k$. So, consider an execution in which more than $n - \ell$ processes decide. Order the processes that decide according to the times when each performs its last scan, and let q_0 be the $(n - \ell + 1)$ th process in this ordering. Let X be the set of at most m different pairs that appear

in the vector that q_0 's final scan returns. Let V be the set of values that appear in pairs of X . Then, $|V| \leq |X| \leq m$. We prove that q_0 and all processes that come later in the ordering output values in V . Thus, the total number of values output is at most $(n - \ell) + |V| \leq n - (n - k + m) + m = k$.

Lemma 4. *In any configuration after q_0 performs its final scan, only pairs with values in V can appear in two or more locations of A .*

Proof. Let C_0 be the configuration just after q_0 's final scan. We shall show by induction that in each configuration reachable from C_0 , only pairs with values in V can appear in two or more locations of A . For the base case, consider the configuration C_0 . By the definition of V , A contains only pairs with values in V , so the claim holds.

For the induction step, suppose the claim holds in all configurations from C_0 to some configuration C_1 reachable from C_0 . Let st be a step that takes the system from C_1 to another configuration C_2 . We show that the claim holds in configuration C_2 . We need only consider the case where st is an update by some process p_{id} . Let (v, id) be the pair that st stores in a component of A .

CASE 1: st is the first update by p_{id} after C_0 . If $v \in V$, then st cannot cause a violation of the claim. If $v \notin V$, then A contains exactly one copy of (v, id) in configuration C_2 , since $(v, id) \notin X$, so again st preserves the claim.

CASE 2: st is not the first update by p_{id} after C_0 . Let s_{id} be the vector obtained by p_{id} 's last scan before st . We show that $v \in V$, and hence st preserves the claim, by considering two subcases.

CASE 2A: s_{id} satisfies the condition on line 0. Then, p_{id} updates its *pref* variable at line 0, so the value v is the value of a pair that appears twice in s_{id} . By the induction hypothesis, $v \in V$.

CASE 2B: s_{id} does not satisfy the condition on line 0. We first argue that at least one pair appears twice in s_{id} . Recall that there are at most $\ell - 1$ undecided processes in C_0 . Since A contains at most m distinct pairs ($|X| \leq m$) in C_0 and at most $\ell - 1$ processes update A after C_0 , Lemma 3 implies that, when the scan s_{id} is performed, A contains at most $m + \ell - 1 = n + 2m - k - 1$ distinct pairs. Since there are $r = n + 2m - k$ locations in A , at least one pair appears twice in s_{id} .

Since q_0 has previously output a value, s_{id} contains no \perp elements. Thus, the reason that s_{id} does not satisfy the condition on line 0 must be that for some j different from p_{id} 's position i , $s_{id}[j] = (pref, id)$. Just before taking the scan s_{id} , p_{id} stores (v, id) in location i . This update occurs after C_0 , since st is not the first update by p_{id} after C_0 . In the configuration after this update of location i , both $s_{id}[j]$ and $s_{id}[i]$ contain (v, id) . So, by the induction hypothesis, $v \in V$. \square

Lemma 4 implies that all processes after the $(n - \ell)$ th in the ordering can only decide one of the (at most) m values in V and, thus, **k -agreement** is ensured.

To prove **m -obstruction-freedom**, consider an execution where the set P of processes that take infinitely many steps has size at most m . To derive a contradiction, assume some process in P never decides. In each loop iteration, a process either keeps its preferred value and increments i (its location to update) modulo r or sets its preferred value without modifying i . We partition P into two subsets: the set NS of “non-stabilizing” processes that modify i infinitely often and the set S of “stabilizing” processes that eventually get stuck updating the same location i forever.

Lemma 5. *There is at least one process in NS .*

Proof. To derive a contradiction, assume the claim is false (i.e., $P = S$). Let μ be a time after which only processes in P take steps and no process changes its local variable i . Then there is a set M of at most m locations whose contents are updated after μ . Let NM be the set of at least

$n + m - k \geq 2$ locations that are not updated after μ . Let μ' be any time when each process in P has performed at least one update after μ . Thus, at μ' , every location in M contains a pair stored by a process in P .

Let p be a process in P that performs a scan that returns a vector s_p after μ' . By the hypothesis, p changes its preferred value in every iteration after μ' , so s_p satisfies the condition on line 0. Process p then changes $pref$ to a value v in a pair (v, k) that appears twice in s_p . Since each component in M is updated by different processes, no two can contain the same pair after μ' . We consider two cases.

CASE 1: in s_p , (v, k) appears in one component of M and one of NM . As (v, k) is read from a component in M after μ' , $p_k \in P$. Consider the time (after μ) at which p_k stores (v, k) in a component in M . Since no register in NM ever changes its value after μ , in p_k 's subsequent scan, (v, k) is in some register of NM and p_k will not change its preferred value, contradicting the fact that $P = S$.

CASE 2: in s_p , (v, k) appears in two components of NM . By the definitions of NM and μ' , (v, k) is found twice in NM at all times after μ . As p changes its preferred value after its next update, it must have found another pair that appears twice and was not in A previously. Then this new pair cannot be in two locations in NM . The pair cannot be in two locations in M either because all the locations of M are updated by different processes. Thus, this new pair is in one location of M and one location of NM . But, as we have seen in Case 1, this leads to a contradiction. \square

Thus, some process updates each component of A infinitely often, yielding the following corollary.

Corollary 6. *There is a time after which A contains only pairs stored by processes in P .*

By Corollary 6, there is a time ν after which (1) A contains only pairs stored by processes in P . By Lemma 3, (2) all pairs in A with the same id have the same value. By the assumption, (3) $|P| \leq m$. (1), (2) and (3) imply that after ν , each time a process $p \in P$ performs a scan it finds at most m different pairs in the snapshot and decides. This contradiction establishes the **m -obstruction-freedom** property.

Theorem 7. *For $1 \leq m \leq k < n$, there is an m -obstruction-free algorithm that solves k -set agreement among n processes using $\min(n + 2m - k, n)$ registers.*

Proof. We established above that the algorithm in Figure 3 solves the problem using a snapshot object of $n + 2m - k$ components. If $n + 2m - k \leq n$, the snapshot object can be implemented from $n + 2m - k$ registers [5]. Otherwise, the snapshot can be implemented from n single-writer registers [1, 13]. \square

4.2 Repeated k -set agreement

The one-shot k -set agreement algorithm can be transformed into an algorithm for repeated set agreement with the same space complexity to prove the following theorem. Since it is quite similar to the one-shot algorithm, we describe it briefly.

The pseudocode for our repeated k -set agreement algorithm is given in Figure 4. It essentially follows the pseudocode of the one-shot algorithm (Figure 3), with additional “shortcuts” which a process may use to adopt a value output previously by another process that has already reached

```

1  Shared variable:
2     $A$ : snapshot object with  $r = n + 2m - k$  components, each initially  $\perp$ 

3  Persistent local variables:
4     $i \leftarrow 0$ 
5     $t \leftarrow 0$ 
6     $history \leftarrow$  empty sequence

7  PROPOSE( $v$ )
8     $t \leftarrow t + 1$ 
9    if  $|history| \geq t$  then
10     output the  $t$ -th value in  $history$  and halt
11     $pref \leftarrow v$ 
12    loop
13      update  $i$ th component of  $A$  with  $(pref, id, t, history)$ 
14       $s \leftarrow$  scan of  $A$ 
15      if  $\exists j$  such that  $s[j] = (w, id', t', his)$  with  $t' > t$  then
16         $history \leftarrow his$ , output the  $t$ -th value in  $his$  and halt
17      if  $|\{s[j] : 0 \leq j < r\}| \leq m$  and  $\forall j, s[j]$  is neither  $\perp$  nor of the form  $(w, q, t', his)$  with  $t' < t$  then
18        let  $j_1 \leftarrow \min\{j_1 : \exists j_2 > j_1 \text{ such that } s[j_1] = s[j_2]\}$ 
19        let  $w$  be value in  $s[j_1]$ 
20         $history \leftarrow history \cdot w$ 
21        output  $w$  and halt
22      if  $\forall j \neq i, s[j] \notin \{\perp, (pref, id, t, history)\}$  and  $\exists j_1 \neq j_2$  such that  $s[j_1]$  and  $s[j_2]$  contain
        identical  $t$ -tuples then
23         $j_1 \leftarrow \min\{j_1 : \exists j_2 > j_1 \text{ such that } s[j_1] \text{ and } s[j_2] \text{ contain identical } t\text{-tuples}\}$ 
24         $pref \leftarrow$  value in  $s[j_1]$ 
25      else  $i \leftarrow (i + 1) \bmod r$ 
26    end loop
27 end PROPOSE

```

Figure 4: Algorithm for m -obstruction-free repeated k -set agreement.

a higher instance of repeated set agreement. Also, a value stored by a process in a lower instance is treated as \perp . Thus, a process decides in instance t only if all tuples found in A are stored by processes in instance t and there are at most m distinct tuples, or if another process has reached an instance higher than t .

Each process p maintains a local variable $history$ that stores a sequence of output values that have been produced in the first instances of repeated k -set agreement. In the current instance t , p essentially follows the one-shot algorithm (Figure 3), except that it appends the current instance number t and $history$ to each value it stores in the shared memory. Thus, each element of the vector returned by a scan of A contains either \perp or a tuple of the form (id, v, t', his) . If $t' > t$, then p_{id} has already completed instance t and his contains the corresponding output value. If this is the case, p adopts all the values output by p_{id} for instances from t to $t' - 1$. If $t' < t$, indicating that

p_{id} has not yet reached instance t , then the position of A is treated as if it were \perp in the one-shot algorithm.

To prove **k -agreement**, we focus on processes that produce their output for instance t without adopting a value from the history that another process stored in A . We call these *t -deciding processes*. Since each other processes that completes its t th PROPOSE adopts one of the value of a t -deciding process, it suffices to prove that t -deciding processes output at most k different values. As in the proof for the one-shot case, we show that the last $\ell = n - k + m$ t -deciding processes output at most m values. There is one complication in the argument: after the $(n - \ell + 1)$ th t -deciding process performs its last scan during instance t , processes may store a t' -tuple with $t' < t$. We show that each process can do this only in a single location, which ensures the agreement property for instance t is not disrupted.

To show **m -obstruction-freedom**, consider an execution where the set P of processes that take infinitely many steps has size at most m . To derive a contradiction, assume some process in P does not complete a PROPOSE. Let t be the smallest number for which some process does not complete its t th PROPOSE and let P' be the set of processes that do not complete their t th PROPOSE. Since the processes in P' never witness the presence of a process in a higher instance of set-agreement, the argument for the one-shot case can be applied to this set P' to obtain the desired contradiction.

A detailed proof of the algorithm can be found in Appendix A.

Theorem 8. *For $1 \leq m \leq k < n$, there is an m -obstruction-free algorithm that solve repeated k -set agreement among n processes using $\min(n + 2m - k, n)$ registers.*

5 Lower Bound for Anonymous One-Shot Agreement

We now turn to anonymous algorithms, where processes are not equipped with identifiers and are programmed identically. We also assume that the domain of possible input values is \mathbb{N} . In this section, we show that any n -process anonymous algorithm for m -obstruction-free (one-shot) k -set agreement requires $\Omega(\sqrt{\frac{nm}{k}})$ registers. Note that this bound on space complexity reflects all three parameters: increasing n or m makes the problem harder and increasing k makes the problem easier. It also generalizes the anonymous result of Fich, Herlihy and Shavit [6] (which is the special case when $m = k = 1$) by showing the dependence on two additional parameters m and k . The assumption of anonymity allows us to add *clones* to an execution. A clone of a process p is another process p' that has the same input as p . Whenever p takes a step, p' takes an identical step immediately afterwards.

Let A be an anonymous algorithm that solves m -obstruction-free k -set agreement among n processes using finitely many registers. For each set V of m distinct input values, fix an execution $\alpha(V)$ such that at most m processes take steps during α and output all values in V . (Such an execution exists, by Lemma 1.) Let $\mathbf{R}(V)$ be the sequence of distinct registers written during $\alpha(V)$ in the order they are first written in $\alpha(V)$. For any sequence \mathbf{R} of distinct registers, define $\mathfrak{V}(\mathbf{R}) = \{V \subset \mathbb{N} : |V| = m \text{ and } \mathbf{R} \text{ is a prefix of } \mathbf{R}(V)\}$.

Lemma 9. *Let $r > 0$ and suppose $n \geq \lceil \frac{k+1}{m} \rceil (m + \frac{r^2-r}{2})$. Then, for $i = 0, \dots, r+1$, there is a sequence \mathbf{R}_i of length i such that $\mathfrak{V}(\mathbf{R}_i)$ is an infinite set.*

Proof. We prove the claim by induction on i .

Base case ($i = 0$): \mathbf{R}_0 is the empty sequence and $\mathbf{v}(\mathbf{R}_0) = \{V \subset \mathbb{N} : |V| = m\}$ is infinite.

Induction step: Let $i \in \{1, 2, \dots, r+1\}$. Assume there is a sequence $\mathbf{R}_{i-1} = \langle R_1, R_2, \dots, R_{i-1} \rangle$ such that $\mathbf{v}(\mathbf{R}_{i-1})$ is infinite.

The induction step is technical, so we begin with an informal overview. Let $c = \lceil \frac{k+1}{m} \rceil$. We first show that there cannot be c disjoint sets V_1, \dots, V_c in $\mathbf{v}(\mathbf{R}_{i-1})$ such that each $\alpha(V_\ell)$ writes only to registers in \mathbf{R}_{i-1} ; otherwise, we could glue together the $\alpha(V_\ell)$'s so that each $\alpha(V_\ell)$ is invisible to all the others, and the number of output values in this glued-together execution would be $|V_1 \cup V_2 \cup \dots \cup V_c| = mc > k$. Then, the rest of the argument is easy: infinitely many sets in $\mathbf{v}(\mathbf{R}_{i-1})$ must have register sequences of length at least i . Since there are only finitely many registers, infinitely many of those sets have the same register R in position i of their sequence. These form the infinite set $\mathbf{v}(\mathbf{R}_i)$, where $\mathbf{R}_i = \mathbf{R}_{i-1} \cdot R$.

To derive a contradiction, assume that (*) there exist c disjoint sets V_1, \dots, V_c in $\mathbf{v}(\mathbf{R}_{i-1})$ such that for all ℓ , $\alpha(V_\ell)$ writes only to registers in \mathbf{R}_{i-1} . Let P_1, \dots, P_c be c disjoint sets of m processes each. The following claim describes how we can glue together the $\alpha(V_\ell)$'s. If β is an execution and P is a set of processes, $\beta|P$ denotes the subsequence of β consisting of steps taken by processes in P .

CLAIM: For $j = 0, 1, \dots, i-1$, there exists an execution β_j with the following properties.

1. Exactly $\frac{cj(j-1)}{2}$ processes outside of $P_1 \cup \dots \cup P_c$ take steps during β_j .
2. For $\ell = 1, \dots, c$, there is a write by some process in P_ℓ to each of R_1, R_2, \dots, R_j during β_j .
3. No process writes to any register outside of $\{R_1, R_2, \dots, R_j\}$ during β_j .
4. For $\ell = 1, \dots, c$, $\beta_j|P_\ell$ is the prefix of $\alpha(V_\ell)$ up to but not including the first write to R_{j+1} (or the entire execution $\alpha(V_\ell)$ if $j = i-1$).

We prove the claim by inductively constructing the executions β_j .

BASE CASE ($j = 0$): We build β_0 by concatenating the maximal prefixes of $\alpha(V_1), \alpha(V_2), \dots, \alpha(V_c)$ that do not contain any writes, performed by process sets P_1, \dots, P_c , respectively. No processes outside $P_1 \cup \dots \cup P_c$ take steps in β_0 . Property 2 is vacuously satisfied. Properties 3 and 4 follow immediately from the definition of β_0 .

INDUCTIVE STEP: Let $j \in \{1, \dots, i-1\}$. Assume that there is a β_{j-1} satisfying the four properties. We describe how to construct β_j .

For each ℓ , we insert $j-1$ clones of processes in P_ℓ , and we pause one clone just before the last write by a process in P_ℓ to each of R_1, \dots, R_{j-1} . Such a write exists, by property 2 of the induction hypothesis. Moreover, there are enough processes to create these clones, since the number of processes that take steps in β_{j-1} plus the $c(j-1)$ additional clones needed to construct β_j total at most $mc + \frac{c(j-1)(j-2)}{2} + c(j-1) = mc + \frac{cj(j-1)}{2} \leq mc + \frac{c(i-1)(i-2)}{2} \leq mc + \frac{cr(r-1)}{2} = \lceil \frac{k+1}{m} \rceil (m + \frac{r^2-r}{2})$ and by the hypothesis of the lemma, there are this many processes in the system.

Let β'_{j-1} be the execution that results from adding all of the clones to β_{j-1} . We add some more steps to the end of β'_{j-1} as follows. For each $\ell = 1, \dots, c$, we add a block write by the clones of processes in P_ℓ followed by steps of processes in P_ℓ continuing the steps of $\alpha(V_\ell)$ until some process is poised to write to R_{j+1} for the first time (or until the end of $\alpha(V_\ell)$ if $j = i-1$). (This is legal, because the block write ensures that all registers have the same state as they would have after $\beta_{j-1}|P_\ell$, which is a prefix of $\alpha(V_\ell)$, by induction hypothesis 4.) Thus, we ensure that β_j satisfies property 4.

By property 4 of the inductive hypothesis, the first newly added step by a process in P_ℓ writes to R_j . Combined with induction hypothesis 2, this proves property 2. For $j < i - 1$, property 3 holds because we stop the processes in P_ℓ just before they write to any register outside of $\{R_1, \dots, R_j\}$. For $j = i - 1$, property 3 follows from our assumption (*) that $\alpha(V_\ell)$ writes only to registers in \mathbf{R}_{i-1} .

The processes outside $P_1 \cup \dots \cup P_c$ that take steps in β_j are the $\frac{c(j-1)(j-2)}{2}$ processes that take steps in β_{j-1} plus the $c(j-1)$ clones that we added when constructing β'_{j-1} . So the total number of such processes is $\frac{cj(j-1)}{2}$, satisfying property 1. This completes the proof of the claim.

In β_{i-1} processes in P_ℓ output all m values in V_ℓ (for all ℓ). Since V_1, \dots, V_c are disjoint sets, there are at least $cm = \lceil \frac{k+1}{m} \rceil \cdot m \geq k+1$ different output values in β_{i-1} . This contradicts the k -agreement property. Thus, assumption (*) is false, so there are fewer than c disjoint sets in $\mathfrak{V}(\mathbf{R}_{i-1})$ such that $\alpha(V_\ell)$ writes only to registers in \mathbf{R}_{i-1} . Thus, there are infinitely many sets V in $\mathfrak{V}(\mathbf{R}_{i-1})$ such that $\alpha(V)$ writes outside of \mathbf{R}_{i-1} . Since there are only finitely many registers, there must be infinitely many of these sets V such that the first register outside of \mathbf{R}_{i-1} written during $\alpha(V)$ is the same for all V . Call that register R . Let \mathbf{R}_i be obtained by concatenating R to the end of \mathbf{R}_{i-1} . Then, there are infinitely many sets V such that \mathbf{R}_i is a prefix of $\mathbf{R}(V)$. This completes the proof. \square

Theorem 10. *Any anonymous algorithm that solves m -obstruction-free k -set agreement among n processes using registers must use more than $\sqrt{m(\frac{n}{k} - 2)}$ registers.*

Proof. Assume an algorithm solves the problem using r registers where $r \leq \sqrt{m(\frac{n}{k} - 2)}$. Then,

$$\begin{aligned}
r &\leq \sqrt{m\left(\frac{n}{k} - 2\right)} \\
&\leq \sqrt{m\left(\frac{2n}{k+m} - 2\right)} \\
&\quad (\text{ since } m \leq k \Rightarrow \frac{2n}{k+m} \geq \frac{n}{k}) \\
&= \sqrt{\frac{2m(n-k-m)}{k+m}} \\
\Rightarrow r^2 - r &\leq \frac{2m(n-k-m)}{k+m} \\
\Rightarrow \frac{k+m}{m} \cdot \frac{r^2-r}{2} &\leq n - k - m \\
\Rightarrow n &\geq \frac{k+m}{m} \left(m + \frac{r^2-r}{2}\right) \\
&\geq \lceil \frac{k+1}{m} \rceil \left(m + \frac{r^2-r}{2}\right).
\end{aligned}$$

So, by Lemma 9 there exists a sequence of $r+1$ registers used in some executions of A , which is impossible since there are only r registers. \square

6 Anonymous Algorithm for Repeated Set Agreement

Theorem 11. *There is an algorithm that solves m -obstruction-free repeated k -set agreement among n processes (for $m \leq k$) using $(m+1)(n-k) + m^2 + 1$ registers.*

The anonymous algorithm presented in Figure 5 solves m -obstruction-free repeated k -set agreement among n processes (for $m \leq k$) using $(m+1)(n-k) + m^2 + 1$ registers. The algorithm uses the same basic idea as the one in Section 4. It uses a snapshot object with $r = (m+1)(n-k) + m^2$ components, which can be built anonymously and non-blocking using r registers [7]. Again, the idea is to allow the first $\ell = n + m - k$ processes to choose arbitrary outputs and then ensure that the last $n - \ell = k - m$ processes output at most m different values, for a total of at most k different values.

For one-shot k -set agreement, processes alternate between storing their preferred value in a component of the snapshot object A and performing a scan of A . The conditions for outputting a value and adopting a new preference differ from the algorithm in Section 4 to compensate for the lack of identifiers. Whenever a process observes m or fewer different values in a scan, it can output the one that occurs most frequently. Otherwise, if a process sees fewer than ℓ copies of its own preference and at least ℓ copies of another value, it adopts this other value as its preference.

The adaptation of this algorithm to repeated consensus is similar to the technique used for the non-anonymous case. There is one additional complication: there is no known space-efficient wait-free anonymous snapshot implementation from registers, so we use a non-blocking implementation. Therefore, some processes may *starve* while accessing the snapshot object, under the condition that at least one process manages to complete infinitely many instances of k -set agreement.

To ensure that starving processes also complete their PROPOSE operations we use one additional register H where “fast” processes write their outputs. Every process periodically checks H in a parallel thread (lines 0-0) and if it finds out that $|H| \geq t$, where t is the instance of agreement the process is working on, it outputs the t -th value found in H . As in the non-anonymous case, the sequence of values that have been output in the instances of repeated k -set agreement the process has completed so far is stored in a local variable *history*. To ensure that *history* is updated exactly once per instance of k -set agreement, we require that the threads of a process are scheduled so that the pairs of lines 0-0, 0-0, and 0-0 are executed without interruption from the process’s other thread.

The proof of correctness of our algorithm is given in Appendix B.

7 Concluding Remarks

A small gap remains between the upper and lower bounds for non-anonymous repeated set agreement. The one-shot algorithm of [4] uses fewer registers than ours for one special case: when $m = 1$ and $k = n - 1$, it uses two registers compared to our three. This suggests the upper bound could perhaps be improved to $n + m - k$. The gaps are larger for the other scenarios shown in Figure 1. It would be interesting to see if there is an anonymous algorithm that uses linear space, rather than quadratic space. Another natural continuation of this work would be to extend the one-shot anonymous lower bound to the non-anonymous setting. However, closing the gap for the one-shot setting eludes us still.

References

- [1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, September 1993.

```

1  Shared variables:
2       $A$ : snapshot object with  $r = (m + 1)(n - k) + m^2$  components, each initially  $\perp$ 
3       $H$ : register, initially the empty sequence

4  Persistent local variables:
5       $i \leftarrow 0$ 
6       $t \leftarrow 0$ 
7       $history \leftarrow$  empty sequence

8  PROPOSE( $v$ )
9      write  $history$  into  $H$ 
10      $t \leftarrow t + 1$ 
11     if  $|history| \geq t$  then
12         output the  $t$ th value in  $history$  and halt
13     run the following two threads in parallel until one of them produces an output
14     thread 1:
15          $pref \leftarrow v$ 
16          $\ell \leftarrow n + m - k$ 
17         loop
18             update  $i$ th component of  $A$  with value  $(pref, t, history)$ 
19              $s \leftarrow$  scan of  $A$ 
20             if  $\exists j$  such that  $s[j] = (w, t', his)$  with  $t' > t$  then
21                  $history \leftarrow his$ 
22                 output the  $t$ -th value in  $his$  and halt
23             if  $|\{s[j] : 0 \leq j < r\}| \leq m$  and every entry of  $s$  is a  $t$ -tuple then
24                 let  $w$  be the most common frequent value in  $s$ 
25                  $history \leftarrow history \cdot w$ 
26                 output  $w$  and halt
27             if  $|\{j : s[j] = (pref, t, *)\}| < \ell$  and  $\exists new$  such that  $|\{j : s[j] = (new, t, *)\}| \geq \ell$  then
28                  $pref \leftarrow new$ 
29                  $i \leftarrow (i + 1) \bmod r$ 
30         end loop
31     thread 2:
32         loop
33             if  $|H| \geq t$  then
34                 let  $w$  be the  $t$ th element of  $H$ 
35                  $history \leftarrow history \cdot w$ 
36                 output  $w$  and halt
37         end loop
38 end PROPOSE

```

Figure 5: Anonymous algorithm for m -obstruction-free repeated k -set agreement.

- [2] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t -resilient asynchronous computations. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 91–100, 1993.
- [3] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.
- [4] Carole Delporte-Gallet, Hugues Fauconnier, Eli Gafni, and Sergio Rajsbaum. Black art: Obstruction-free k -set agreement with $|MWMR\ registers| < |processes|$. In *Proc. 1st International Conference on Networked Systems*, volume 7853 of *LNCS*, pages 28–41, 2013.
- [5] Faith Ellen, Panagiota Fatourou, and Eric Ruppert. Time lower bounds for implementations of multi-writer snapshots. *Journal of the ACM*, 54(6), December 2007.
- [6] Faith Ellen Fich, Maurice Herlihy, and Nir Shavit. On the space complexity of randomized synchronization. *Journal of the ACM*, 45(5):843–862, September 1998.
- [7] Rachid Guerraoui and Eric Ruppert. Anonymous and fault-tolerant shared-memory computing. *Distributed Computing*, 20(3):165–177, October 2007.
- [8] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.
- [9] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proc. 23rd International Conference on Distributed Computing Systems*, pages 522–529, 2003.
- [10] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, November 1999.
- [11] Michael Saks and Fotios Zaharoglou. Wait-free k -set agreement is impossible: The topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449–1483, 2000.
- [12] Gadi Taubenfeld. Contention-sensitive data structures and algorithms. In *Proc. 23rd International Symposium on Distributed Computing*, volume 5805 of *LNCS*, pages 157–171, 2009.
- [13] Paul M. B. Vitányi and Baruch Awerbuch. Atomic shared register access by asynchronous hardware. In *Proc. 27th Symposium on Foundations of Computer Science*, pages 233–243, 1986.
- [14] Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proc. 17th ACM Symposium on Principles of Distributed Computing*, pages 297–306, 1998.

A Proof of correctness of repeated set agreement

In this section, we prove Theorem 8. The pseudocode for our repeated k -set agreement algorithm appears in Figure 4. It essentially follows the pseudocode of the one-shot algorithm (Figure 3), with additional “shortcuts” which a process may use to adopt a value output previously by another process that has already reached a higher instance of repeated set agreement. Also, a value stored by a process in a lower instance is treated as \perp . Thus, a process decides in instance t only if all tuples found in A are stored by processes in instance t and there are at most m distinct tuples, or if another process has reached an instance higher than t . The local variable *history* initially stores an empty sequence and the local variable t is initially 0. The local variable i stores the location that the process updates and is initially 0. The values of these three local variables persist from one invocation of PROPOSE to the next. In particular, this means that the first location of a PROPOSE is the last location of the previous PROPOSE.

A process updates components of the shared snapshot object with tuples of the form $(v, id, t, history)$, where v is the process’s preferred value, id is the identifier of the process, t indicates which instance of set agreement the process is currently working on, and *history* is a sequence of output values for instances of set agreement. We refer to a tuple whose third element is t as a t -tuple.

To see that the algorithm satisfies **validity**, first observe that when a process invokes PROPOSE for the t th time, the length of its *history* variable is at least $t - 1$. The value in every t -tuple in A and, thus, put in the t th position of a process’s local variable *history*, is the input value of some process’s t th invocation of PROPOSE.

The following Lemma reformulates Lemma 3 for t -tuples, showing that A cannot contain more than one distinct t -tuple for a given process.

Lemma 12. *Let id be a process identifier and t be a positive integer. In any reachable configuration, all t -tuples with identifier id in A are identical.*

Proof. To derive a contradiction, assume that in some reachable configuration C , $A[i_1] = (v_1, id, t, his_1)$ and $A[i_2] = (v_2, id, t, his_1)$ such that $(v_1, his_1) \neq (v_2, his_1)$. Let p_{id} be the process with identifier id . By the algorithm, p_{id} changes its *history* variable only when it switches to a higher instance of repeated agreement. Thus, $his_1 = his_2$ and we must have $v_1 \neq v_2$. Let C be reached in some execution at time μ . Let u_1 and u_2 be the last update steps before μ in which p_{id} updates $A[i_1]$ and $A[i_2]$, respectively. Without loss of generality, assume that u_1 occurred before u_2 . Then, at some time between u_1 and u_2 , p_{id} changes its *pref* variable in instance t (at line 0). Consider the first time after u_1 when p_{id} performs such a change, and let i^* and s^* be the values of p_{id} ’s local variables i and s at that time. Since (1) $A[i_1] = (v_1, id, t, his_1)$ at all times between u_1 and μ and (2) s^* is obtained between u_1 and μ , $s^*[i_1]$ must be equal to (v_1, id, t, his_1) . By the algorithm, $i^* = i_1$; otherwise, the test in line 0 would not be satisfied, and p_{id} would not change *pref* in line 0. Therefore, in the next iteration of the loop, p_{id} will update location $A[i_1]$. This update is after u_1 and no later than u_2 (and hence before μ), which contradicts the definition of u_1 as the last update performed by p_{id} to $A[i_1]$ before μ . \square

To show **k -agreement**, we use arguments similar to the proof for the one shot algorithm. Let $\ell = n - k + m$. We call a process t -deciding if it outputs a value at line 0 (i.e., without adopting a value from another process’s *history* value) during its t th invocation of PROPOSE. If, for a given instance t , at most $n - \ell$ processes are t -deciding, then k -agreement for instance t is immediate

since $n - \ell = k - m < k$. Otherwise, consider an execution in which more than $n - \ell$ processes are t -deciding. Order these processes according to the time that they perform their last scan in instance t , and let q_0 be the $(n - \ell + 1)$ th process in this ordering. Let X be the set of at most m different tuples that appear in q_0 's final scan and V be the set of values in X . Then, $|V| \leq |X| \leq m$. We shall show that q_0 and all processes that come later in the ordering output values in V . Thus, the total number of output values in instance t is at most $(n - \ell) + |V| \leq n - (n - k + m) + m = k$.

Lemma 13. *After q_0 performs its final scan in instance t , only t -tuples with values in V can appear twice in A .*

Proof. This proof is analogous to the proof of Lemma 4 for the one-shot algorithm. Let C_0 be the configuration just after q_0 's last scan. We shall show by induction that each configuration reachable from C_0 , only t -tuples with values in V can appear in two or more locations of A . For the base case, consider the configuration C_0 . By the definition of V , A contains only tuples with values in V , so the claim holds.

For the induction step, suppose the claim holds in all configurations from C_0 to some configuration C_1 reachable from C_0 . Let st be a step that takes the system from C_1 to another configuration C_2 . We must show that the claim holds in configuration C_2 . We need only consider steps st in which some process p_{id} stores a tuple (v, id, t, his) in A .

CASE 1: st is the first time p_{id} stores a t -tuple after C_0 . If $v \in V$, then st cannot cause a violation of the claim. If $v \notin V$, then A contains exactly one copy of (v, id, t, his) in configuration C_2 , so again st preserves the claim.

CASE 2: st is not the first time p_{id} stores a t -tuple after C_0 . Let s_{id} be the vector obtained by p_{id} 's last scan (at line 0) before st . Since s_{id} is not in the last iteration of the loop during instance t , s_{id} must not satisfy the conditions on line 0 or 0. We show that $v \in V$, and hence st preserves the claim, by considering two subcases.

CASE 2A: s_{id} satisfies the condition on line 0. Since the condition on line 0 is not satisfied and the condition on line 0 is satisfied, every tuple in s_{id} is a t -tuple. Then, p_{id} updates its *pref* variable at line 0, so the value v is the value of a t -tuple that appears twice in s_{id} . By the induction hypothesis, $v \in V$.

CASE 2B: s_{id} does not satisfy the condition on line 0.

We call an update after C_0 *bad* if it stores either a t' -tuple with $t' < t$ or a t -tuple that is not in X . We first argue that each process can do bad updates to at most one location. To derive a contradiction, suppose some process does bad updates to two different locations after C_0 . Consider the first process p to do a bad update to a second location. Process p 's last bad update to one location and its first bad update to the second location must be in the same instance of PROPOSE, because p must execute line 0 between them. Let s_p be the vector returned by the scan that p performs at line 0 during the iteration of the loop when it executes line 0. Then, s_p must not satisfy the conditions on line 0 or 0. Recall that at least $n - \ell + 1$ processes have updated A for the last time during instance t prior to C_0 . So at most $\ell - 1$ processes can do bad updates. Since no process has done bad updates to two locations before the p 's scan obtained the vector s_p , and no location of s_p contains a tuple with instance number greater than t , at least $r - \ell + 1 = m + 1$ locations of s_p contain t -tuples in X . Since $|X| \leq m$, at least two locations of s_p contain the same t -tuple. This contradicts the fact that s_p does not satisfy the condition on line 0. Thus, each process can do bad updates to at most one location.

Hence, at all times after C_0 , at least $r - (\ell - 1) = m + 1$ locations have not had any bad updates performed on them. Since s_{id} did not satisfy the condition on line 0, s_{id} must contain at least

$m + 1$ t -tuples in X , and therefore s_{id} contains at least two identical t -tuples. Moreover, some process q_0 satisfied the condition on line 0 prior to the scan that returned s_{id} , so no component of s_{id} contains \perp . Thus, the only reason s_{id} does not satisfy the condition on line 0 must be that for some j different from p_{id} 's position i , $s_{id}[j] = (v, id, t, his)$. Just before taking the scan s_{id} , p_{id} updates location i with (v, id, t, his) . This update occurs after C_0 , since st is not the first update by p_{id} after C_0 . In the configuration after this update to location i , both $s_{id}[j]$ and $s_{id}[i]$ contain (v, id, t, his) . So, by the induction hypothesis, $v \in V$. \square

Lemma 13 implies that all t -deciding processes after the $(n - \ell)$ th output values in V and, thus, a total of at most $n - \ell + m = k$ values are output by t -deciding processes. The **k -agreement** property follows.

To prove **m -obstruction-freedom**, consider an execution where the set P of processes that take infinitely many steps has size at most m . To derive a contradiction, assume that some process in P completes only a finite number of PROPOSE operations. Let t be the smallest number such that a process in P does not complete its t th PROPOSE. Let P' be the set of processes in P that do not complete the t th PROPOSE. By the algorithm, no process in P' ever witnesses the presence of a process in a higher instance; otherwise, it would output a value decided in instance t at line 0.

Eventually, processes stop storing tuples with instance numbers $t' < t$ in A . Below we reuse the arguments of the proof of Lemma 5 to show that at least one process in P' updates each component of A infinitely often.

Recall that each time a process in P' executes the loop in instance t , it either keeps its preferred value and increments i (the next location to update) modulo r or changes its preferred value without modifying i . Let NS denote the set of processes in P' that increment i infinitely often and the set S denotes the rest of the processes in P' , i.e., the processes that eventually get stuck updating to the same location forever.

Lemma 14. $NS \neq \emptyset$.

Proof. The proof is by contradiction. Assume it is not the case ($P' = S$).

Let M be the set of at most m locations that processes in S eventually settle on. Note that no process in $P - P'$ can update a location outside of M infinitely often because then the processes in P' would eventually see a tuple with instance number greater than t and complete their t th PROPOSE operation. Let μ be a time after which only processes in P take steps and no process updates a location outside of M . Let NM be the set of at least $n + m - k \geq 2$ locations that are never changed after μ .

Since all positions in NM that contain tuples of earlier instances are ignored, we simply reuse the arguments of the proof of Lemma 5, to derive a contradiction. \square

By Lemma 14, (1) there is a time after which only tuples stored by processes in P' are found in scans performed by processes in P' , and all of them are t -tuples. By Lemma 12, (2) all t -tuples in A of the same process are identical and (3) $|P'| \leq |P| \leq m$. (1), (2) and (3) imply that there is a time after which, whenever a process $p \in P'$ performs a scan, it finds at most m different t -tuples in the returned vector and, thus, decides, contradicting the definition of P' . This completes the proof of the **m -obstruction-freedom** property.

Thus, we have shown that the algorithm solves repeated k -set agreement using a snapshot object with $n + 2m - k$ registers, which can be implemented using $\min(n, n + 2m - k)$ registers, as described in the proof of Theorem 7. This completes the proof of Theorem 8.

B Proof of correctness of anonymous repeated set agreement

To prove Theorem 11, consider our algorithm in Figure 5. The algorithm actually uses a non-blocking snapshot object with $r = (m+1)(n-k) + m^2$ components, which can be built anonymously using r registers [7], plus one additional register. Each component of the snapshot object is initially \perp .

In this algorithm, a process stores tuples of the form $(v, t, \text{history})$ where v is the process's preferred value, t indicates which instance of set agreement the process is currently working on, and history is a sequence of output values for instances of set agreement. We refer to a tuple whose second element is t as a t -tuple.

As an invariant, it is easy to see that each of the following can only store input values of some process's t invocation of PROPOSE:

- a process's *pref* variable during the process's t th invocation of PROPOSE,
- the first component of a t -tuple appearing in A , and
- the t th element of any sequence that is stored in a process's *history* variable, in the shared variable H or inside a tuple in A .

Validity follows.

Next, we prove the k -**agreement** property. A process is t -*deciding* if it outputs a value on line 0. Any other process that produces an output for its t th PROPOSE operation outputs the same result as some t -deciding process, so it suffices to show that the t -deciding processes output at most k different values. As in Section 4, we show that the last $\ell = n - k + m$ t -deciding processes output at most m different values, so that the total number of outputs for instance t is at most $n - \ell + m = k$ values.

If at most $n - \ell$ processes are t -deciding, then k -agreement is trivial for the t th instance of set agreement, since $n - \ell = k - m < k$. So, consider an execution in which more than $n - \ell$ processes are t -deciding. Order the t -deciding processes according to the time that they perform their last scan in their t th invocations of PROPOSE, and let q_0 be the $(n - \ell + 1)$ th process in this ordering. Let X be the set of tuples that appear in q_0 's final scan. Let V be the values that appear in tuples in X . We prove that q_0 and all t -deciding processes that come later in the ordering output values in V .

We call an update of A after C_0 a *bad update* if it stores a t' -tuple with $t' < t$ or a t -tuple whose value is not in V .

Lemma 15. *After q_0 performs its final scan in its t th PROPOSE operation, each process performs bad updates to at most one component of A .*

Proof. To derive a contradiction, assume that some process performs bad updates to two components of A after q_0 's final scan scan_0 . Consider the first process p to do a bad update on a second location. Let s_p be the vector returned by the last scan that p performs before its bad update to the second location. This scan causes p to execute line 0 so that it can perform an update on the second location. Thus s_p does not contain any t' -tuple with $t' > t$. Since $n - \ell + 1$ processes have performed their final scan of their t th PROPOSE operation at or before scan_0 , at most $\ell - 1$ processes can perform updates that store t' -tuples with $t' \leq t$ after scan_0 . By definition of p , none of those $\ell - 1$ processes have performed bad updates on two different locations between scan_0 and

p 's scan that returned s_p . Since $scan_0$ returned a vector that contained only t -tuples, s_p must contain at most $\ell - 1$ components that are either t' -tuples with $t' < t$ or t -tuples with values not in V . So there are at least $r - \ell + 1 = (m + 1)(\ell - 1) + 1 - (\ell - 1) = m(\ell - 1) + 1$ locations of s_p that contain t -tuples with values in V . Since $|V| \leq m$, one of the values in V must appear in t -tuples stored in at least ℓ locations. Thus p must adopt a value in V after it obtains the scan s_p , contradicting the fact that p 's next update after this scan uses a value not in V . \square

It follows that at any time after q_0 's final scan, there are at most $\ell - 1$ t -tuples in A with values that are not in V . Any t -deciding process ordered after q_0 performs a final scan that returns only t -tuples, so one of the values in V must appear in at least ℓ of them, and is therefore the most frequent value in the scan. Thus, the value output by any such process must be in V . Hence, the total number of values output is at most $(n - \ell) + |V| \leq n - (n - k + m) + m = k$, ensuring that k -agreement is satisfied.

Finally, we prove **m -obstruction-freedom**. For this part of the proof, it is convenient to consider lines 0 to 0 to be a single atomic action. Since there is only one shared-memory access in this block of code, there is no loss of generality in this assumption: every execution has an equivalent execution where this block is executed atomically, so if we prove m -obstruction-freedom for executions that satisfy this assumption then it also holds for all executions.

Consider an execution where at most m processes continue to take steps forever. Let P be the set of processes that complete infinitely many accesses to the snapshot object. P is non-empty, since the snapshot implementation we use is non-blocking, and $|P| \leq m$. To derive a contradiction, assume that some process in P never completes one of its PROPOSE operations. Let t be the smallest number such that some process in P does not complete its t th PROPOSE. Let P' be the set of processes in P that do not complete their t th PROPOSE operation. Let μ be a time after

- every process outside P has stopped performing updates on A ,
- every process in P' has begun its t th PROPOSE operation,
- every process in $P - P'$ has begun its $(t + 1)$ th PROPOSE operation, and
- no component of A contains a t' -tuple for any $t' < t$.

It is possible to choose μ to satisfy the last condition because each process in P' completes infinitely many iterations of the loop and therefore updates every location of A after μ . Thus, eventually all t' -tuples with $t' < t$ are overwritten. Note that after μ , no component of A ever contains a t' -tuple with $t' < t$.

We say that a value v is a *candidate* in a configuration C if it is either the *pref* value of some process in P' or it appears in a t -tuple in A . We shall prove that there is a configuration after μ with at most m candidates. After that point, only those m values can appear in t -tuples in the snapshot object. It follows that every process in P' completes its t th PROPOSE when it next performs a scan, which contradicts the definition of P' .

Lemma 16. *If, in some configuration C after μ , a value v is not the *pref* of any process in P' and t -tuples with value v appear in fewer than ℓ components of the snapshot object, then after some time, v will not be a candidate anymore.*

Proof. To derive a contradiction, assume that some process in P' changes its local $pref$ variable to v in some step after C . Consider the first such step by any process after C . Let $scan$ be the scan performed in that step. Between C and $scan$, no process executing its t th PROPOSE stores a t -tuple with value v , so the result of $scan$ contains t -tuples with value v in at most $\ell - 1$ components, contradicting the fact that p adopts the value v in the step when it performs $scan$.

Thus, no process in P' ever has v as its preferred value after C . So, no t -tuple with value v is ever stored in A after C . Since each process in P' executes infinitely many steps of its t th PROPOSE operation, and increments its index i in every iteration of the loop, it eventually overwrites every component of A . Thus, there is a time (after C), after which no component of A contains a t -tuple with value v . After this time, v is never a candidate. \square

Lemma 17. *Whenever a process in P' performs a scan after μ , there is some value v that appears in t -tuples in at least ℓ components of A .*

Proof. To derive a contradiction, suppose there is no such value v . Consider the configuration C immediately after the scan. By Lemma 16, only the values stored in $pref$ variables of processes in P' remain candidates forever. There are at most m such values. Thus, there is a time after which every t -tuple in A contains only those values. Whenever a process in P' performs a scan after that time, it will terminate, contradicting our assumption that no process in P' ever completes its t th PROPOSE. \square

For any configuration C and value v , let $mult(C, v)$ be the number of components of A that contain t -tuples with value v in C plus the number of poised processes that are poised to perform an update and have $pref v$ in C . The following lemma generalizes Lemma 16.

Lemma 18. *Consider a value v . If, in some configuration C after μ , $mult(C, v) < \ell$, then after some time, v will no longer be a candidate.*

Proof. We first show that if a single step st takes the system from a configuration C_1 to another configuration C_2 and $mult(C_1, v) < \ell$ then $mult(C_2, v) < \ell$. If st is a step by a process in $P - P'$, it can only decrease $mult$. If st is an update by a process in P' , st may increase by one the number of components of A containing a t -tuple with value v , but then st will also decrease the number of processes poised to store a t -tuple with value v by one, so the value of $mult$ cannot be increased by st . Finally, suppose st is an atomic execution of lines 0–0. In C_1 , fewer than ℓ components of A contain t -tuples with value v (since $mult(v, C_1) < \ell$). Moreover, by Lemma 17, there is a value v' such that t -tuples with value v' appear in at least ℓ components of the scan performed during st . Thus, the process performing st adopts some value different from v as its $pref$. So, st cannot increase $mult$ for v .

Thus, in every configuration reachable from C , $mult(C, v) < \ell$. As argued above, any process in P' that performs a scan after C adopts a value different from v . Thus, eventually, no process will have its $pref$ equal to v , and at that time, v will be in at most $\ell - 1$ components of A , so Lemma 16 ensures that v will eventually cease to be a candidate. \square

Now, consider a configuration C immediately after some process has performed an update (after μ). There are $(m + 1)(\ell - 1) + 1$ registers and at most $m - 1$ processes in P' poised to perform an update. Thus, $\sum_v mult(v, C) \leq (m + 1)\ell - 1$. Therefore, at most m values have $mult(C, v) \geq \ell$. All other values will eventually cease to be candidates, by Lemma 18, so eventually there will be at

most m candidates. All processes in P' will then terminate when they next perform a scan, which contradicts our definition of P' .

Thus, we have shown that every process in P completes infinitely many PROPOSE operations. There remains one more thing to show. There may be some processes not in P that takes infinitely many steps. (These are processes that starve in the non-blocking implementation of the snapshot object.) We must show that each such process p also completes all of its PROPOSE operations. Processes in P write longer and longer sequences to H infinitely often and processes not in P eventually stop writing to H . Thus, for all t , p will eventually see a sequence in H of length at least t , and will then complete its t th PROPOSE operation.

This completes the proof of Theorem 11. We remark that for the one-shot case, the register H is not required, so we can solve the one-shot version using one less register.